DYNAMIC PROGRAMMING FOR ACTION IN THE ENVIRONMENT

Cognitive Dynamic Systems, *Simon Haykin*

Presented by

Joo Yeon Kim & Yunseong Lee

THE BIG PICTURE

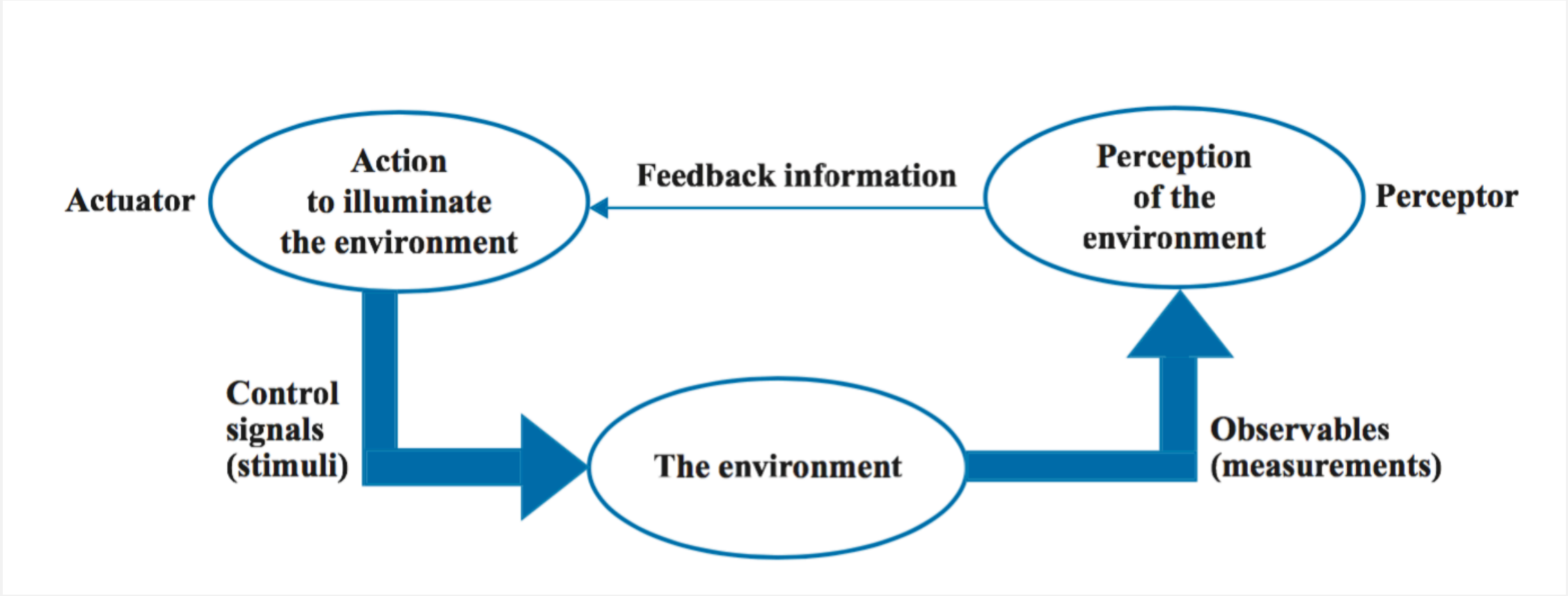
COGNITIVE DYNAMIC SYSTEMS

Build up rules of behavior over time through

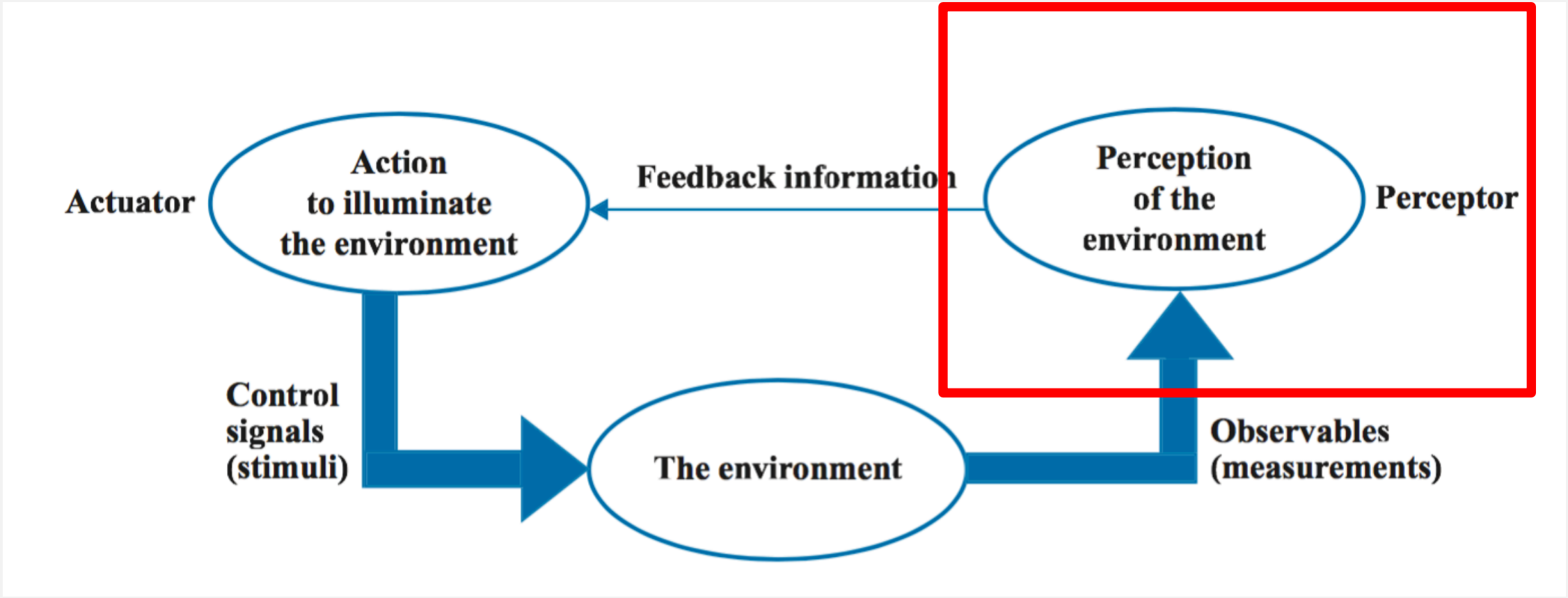
learning from continuous experiential interactions with the environment,

and thereby deal with environmental uncertainties.

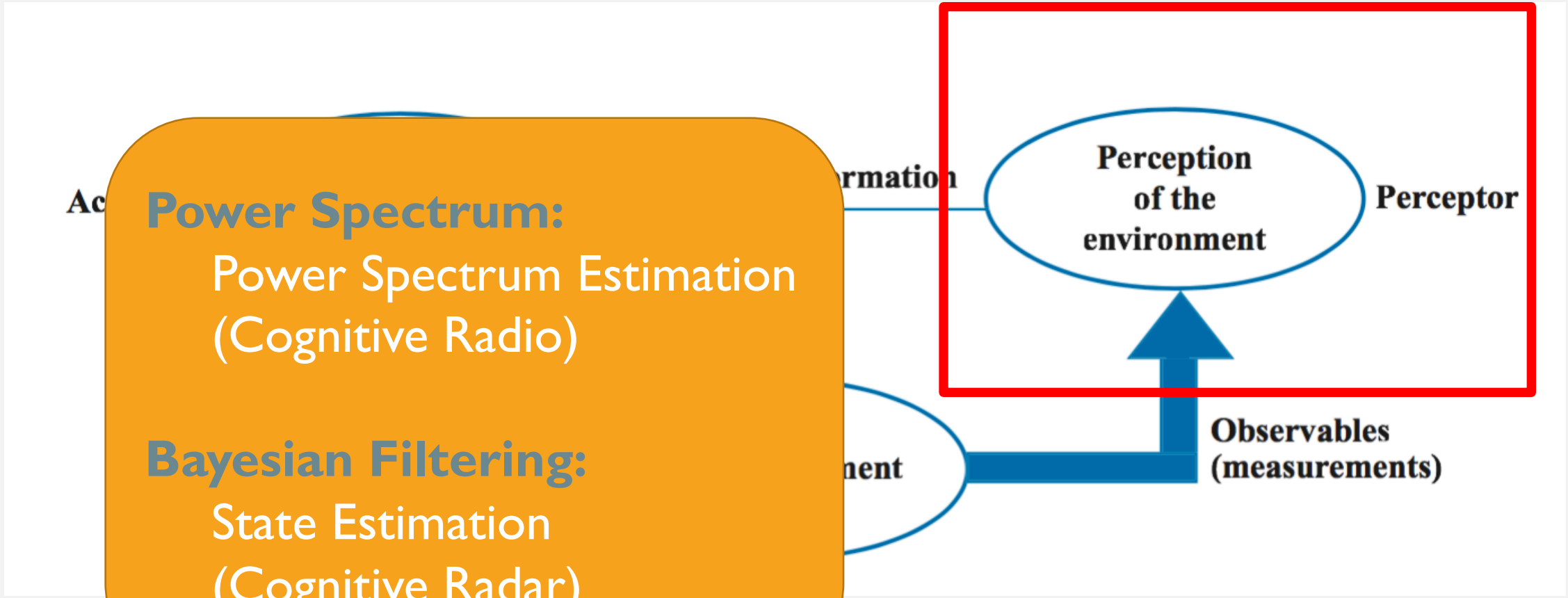
THE PERCEPTION-ACTION CYCLE



THE PERCEPTION-ACTION CYCLE



THE PERCEPTION-ACTION CYCLE



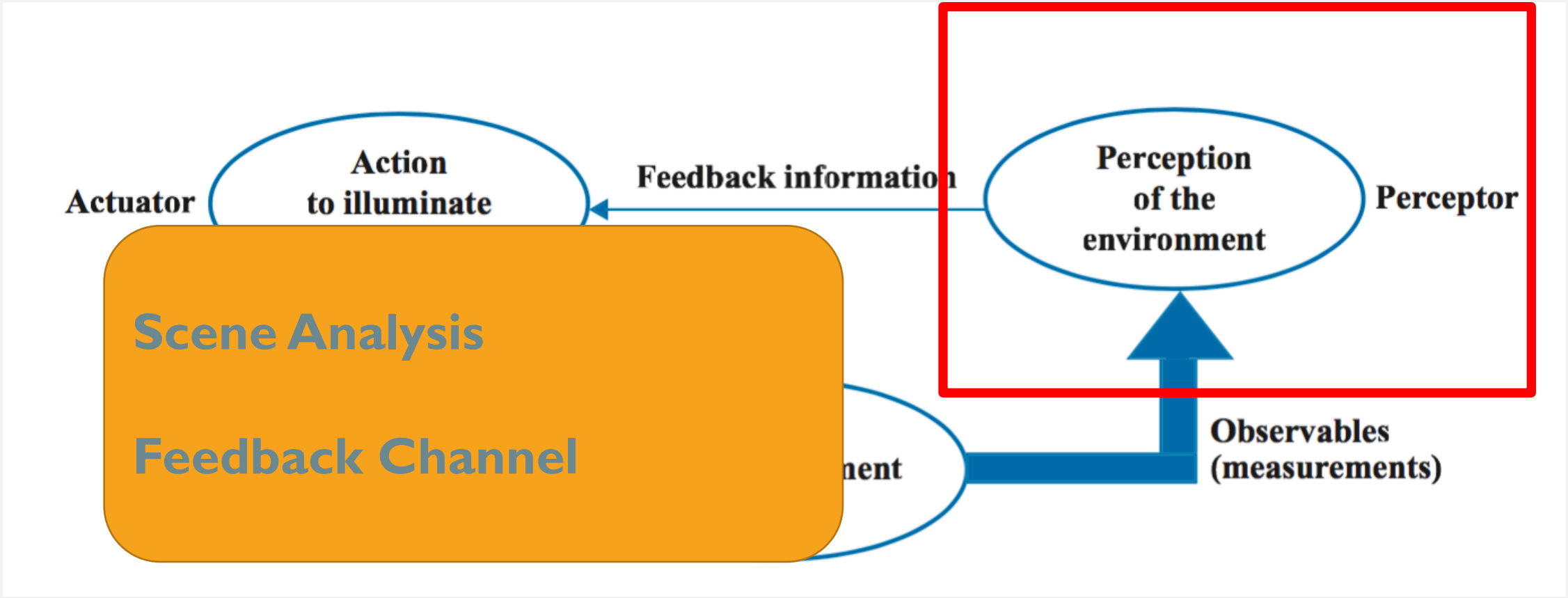
Power Spectrum:

Power Spectrum Estimation
(Cognitive Radio)

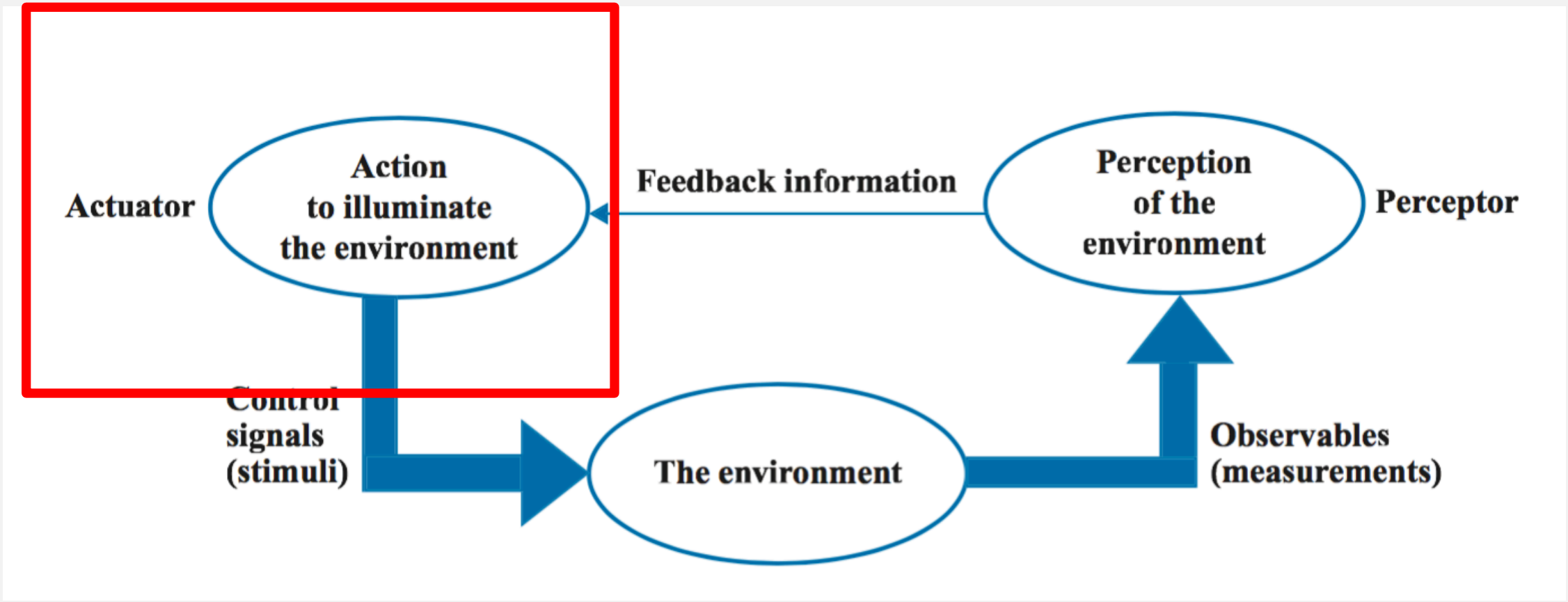
Bayesian Filtering:

State Estimation
(Cognitive Radar)

THE PERCEPTION-ACTION CYCLE



THE PERCEPTION-ACTION CYCLE



DYNAMIC PROGRAMMING

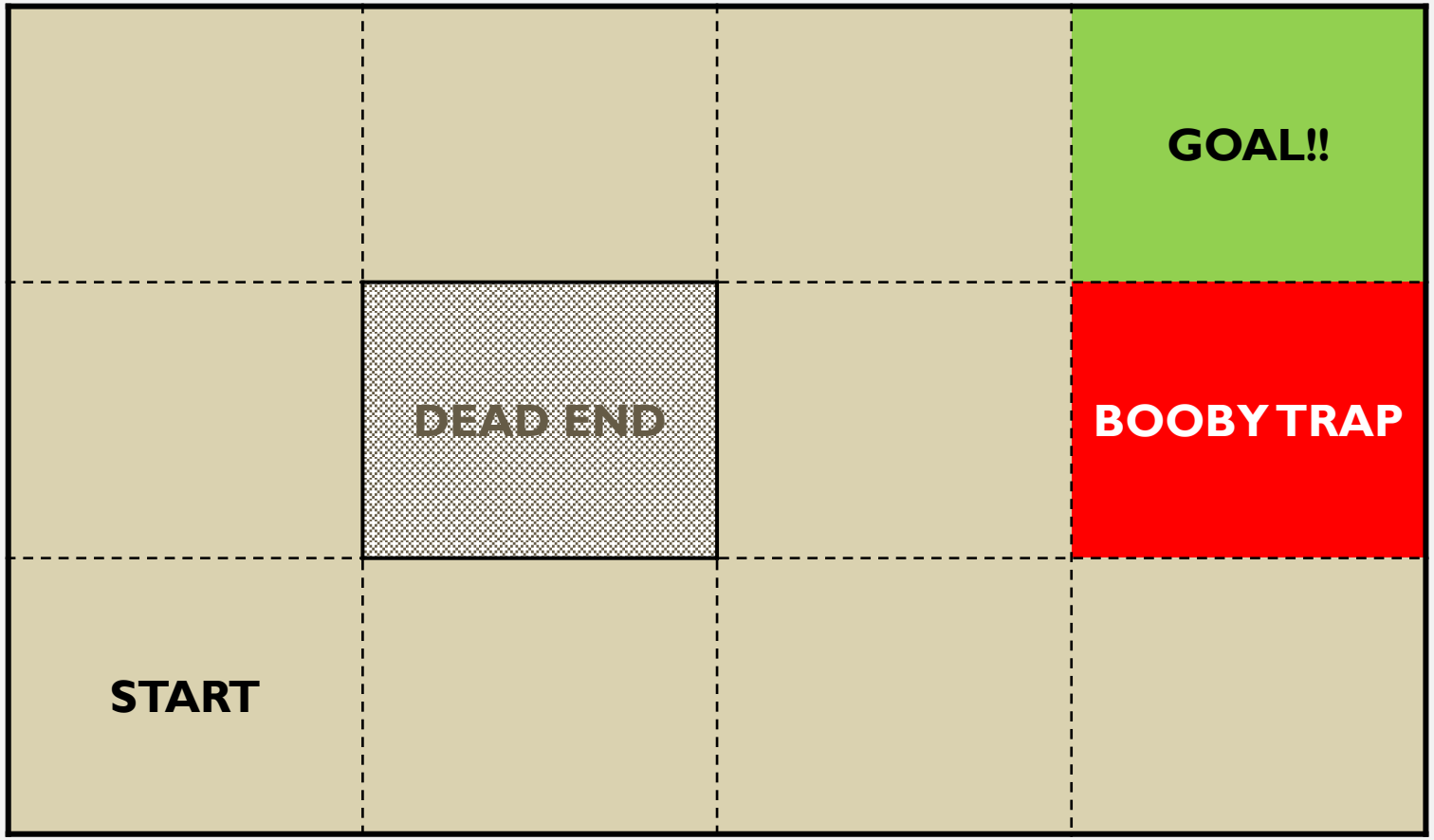
WHAT IS DYNAMIC PROGRAMMING?

- *Dynamic programming* is a technique that deals with situations where decisions are made in stages (i.e. different time steps), with the outcome of each decision being *predictable* to some extent before the next decision is made.
- A key aspect of such situations is that decisions cannot be made in isolation. Rather, the desire for a low cost at the present is balanced against the undesirability of a high cost in the future.

DYNAMIC PROGRAMMING

- Markov Decision Processes
- Bellman’s Optimality Criterion
- Policy Iteration
- Value Iteration

MARKOV'S DECISION PROCESS



You can move UP, DOWN, LEFT, RIGHT

DETERMINISTIC VS. STOCHASTIC

Deterministic

- Every state can be uniquely determined with model parameters and previous states.
- Always perform identically for a given set of initial conditions.

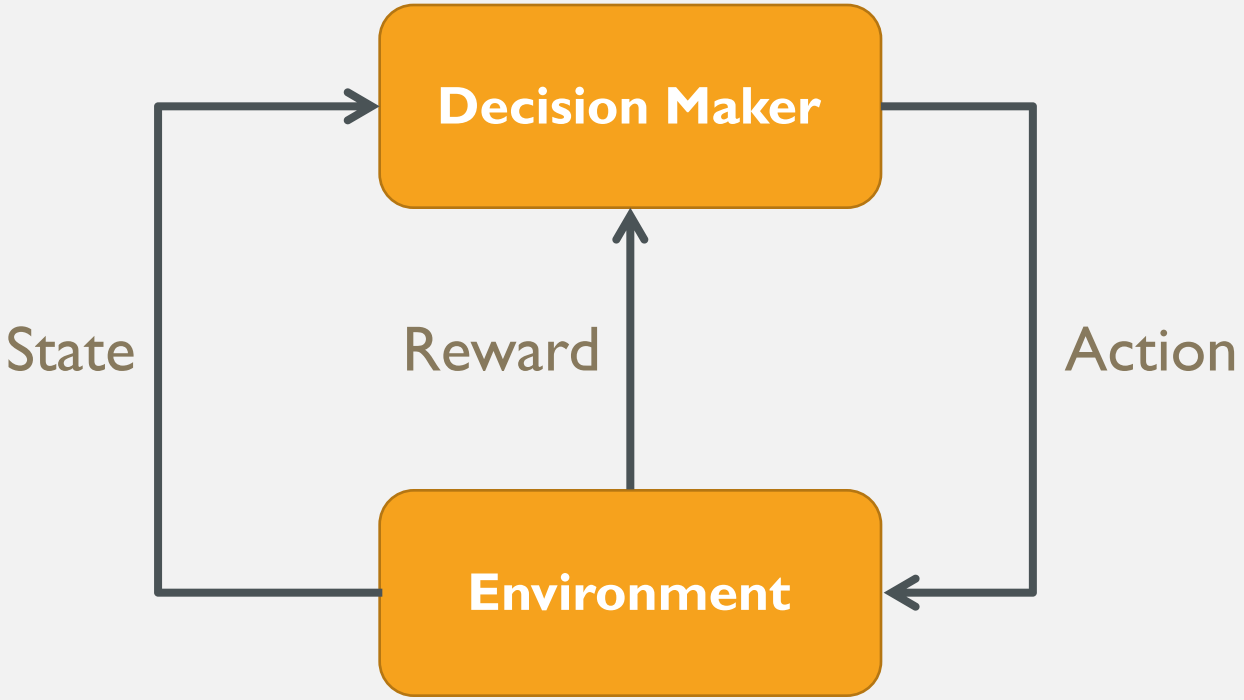
Stochastic

- States determined by probability distributions.
- Such probability distributions include a notion of randomness.

MARKOV'S DECISION PROCESSES

The Actuator's Decision Making Process

Discrete-time & Stochastic



- Set of States: $\boldsymbol{S} = \{\boldsymbol{s}\}$
- Set of Actions: $\boldsymbol{A} = \{\boldsymbol{a}\}, \boldsymbol{a}: \boldsymbol{S} \rightarrow \boldsymbol{S}$
- Reward Function: $\boldsymbol{R}(\boldsymbol{s})$
- Transition Model:

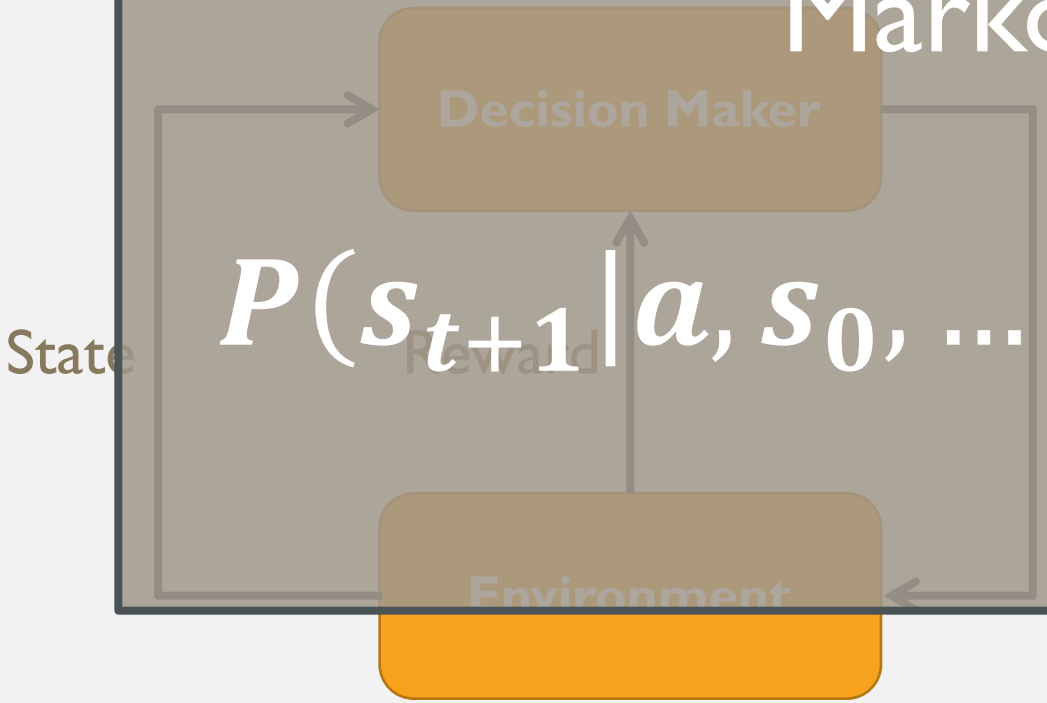
  $\boldsymbol{T}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')$ or $\boldsymbol{P}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$
- Policy: $\boldsymbol{\pi}(\boldsymbol{s}) \rightarrow \boldsymbol{a}$

MARKOV'S DECISION PROCESSES

The Actuator's Decision Making Process

Discrete-time & Stochastic

Markov Property



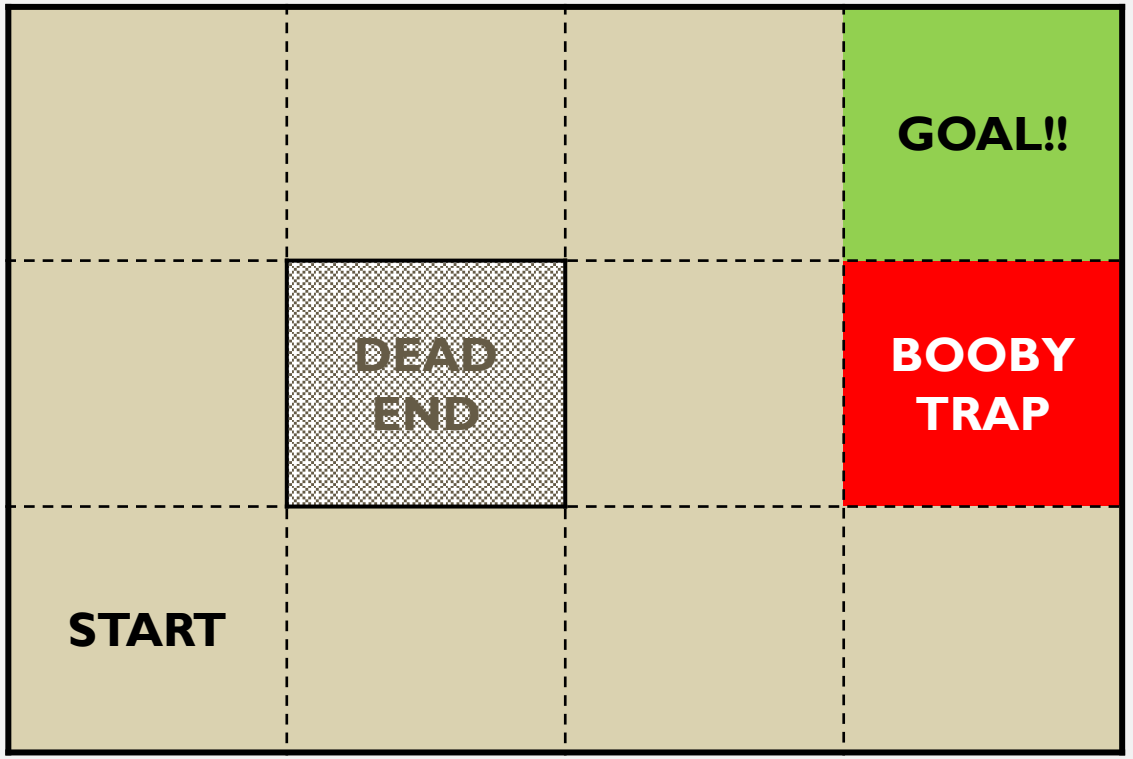
$$P(s_{t+1}|a, s_0, \dots, s_t) = P(s_{t+1}|a, s_t)$$

- Set of States: $S = \{s\}$
- Set of Actions: $A = \{a\}, a: S \to S$
- Reward Function: $R(\cdot)$
- Discount Factor: γ
- Initial State: s_1

MARKOV'S DECISION PROCESS

- Set of States: $\boldsymbol{S = \{s\}}$
- Set of Actions: $\boldsymbol{A = \{a\}, a: S \rightarrow S}$
- Reward Function: $\boldsymbol{R(s)}$
- Transition Model:

  $\boldsymbol{T(s, a, s')}$ or $\boldsymbol{P(s'|s, a)}$
- Policy: $\boldsymbol{\pi(s) \rightarrow a}$



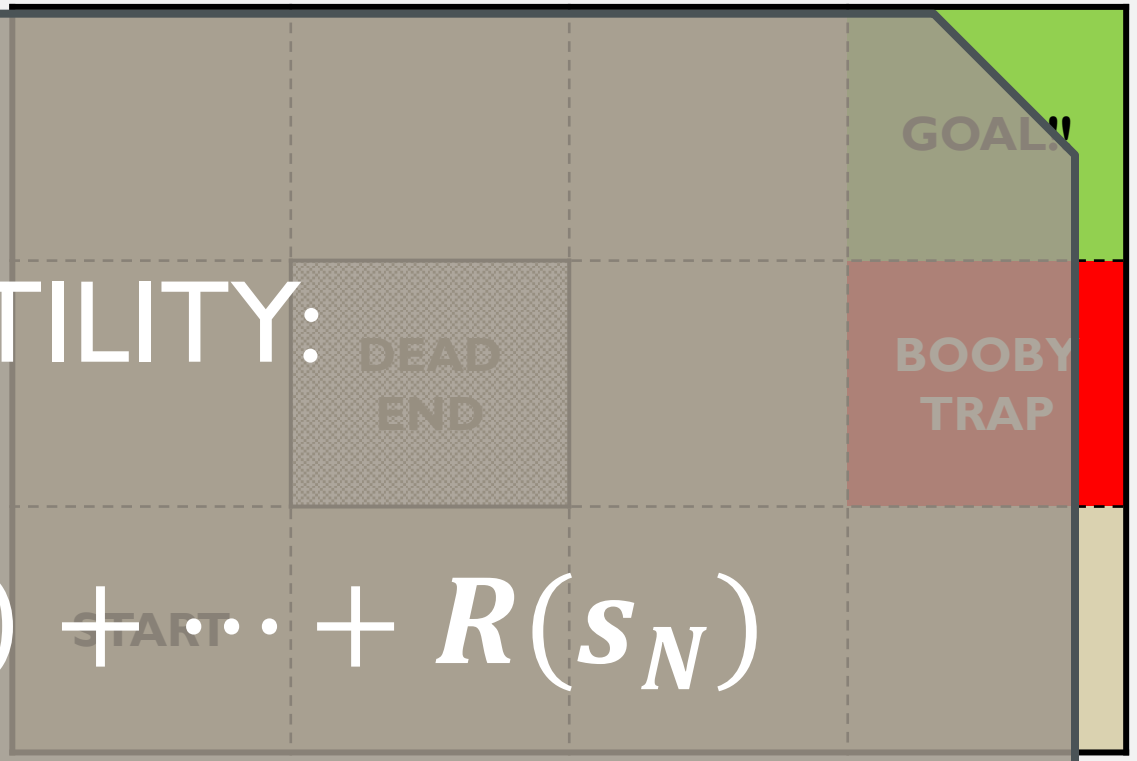
You can move UP, DOWN, LEFT, RIGHT

MARKOV'S DECISION PROCESS

- Set of States: $S = \{s\}$
- Set of Actions: $A = \{a\}, a: S \rightarrow S$
- Reward Function: $R(s)$
- Transition Model:
  $T(s, a, s')$ or $P(s' | s, a)$
- Policy: $\pi(s) \rightarrow a$

Define UTILITY:

$$U(s_0, \dots, s_N) = R(s_0) + R(s_1) + \cdots + R(s_N)$$



You can move UP, DOWN, LEFT, RIGHT

MARKOV'S DECISION PROCESS

$$U(s_0, \dots, s_N) = R(s_0) + R(s_1) + \cdots + R(s_N)$$

But this **N** can increase infinitely!

$$U(s_0, \dots) = R(s_0) + \gamma^1 R(s_1) + \cdots + \gamma^N R(s_N) + \cdots$$

Discount Factor: $0 < \gamma < 1$

MARKOV'S DECISION PROCESS

$$\boldsymbol{U(s_0, \dots) = R(s_0) + \gamma^1 R(s_1) + \cdots + \gamma^N R(s_N) + \cdots}$$

We move from s_t to s_{t+1} by **action** $\boldsymbol{\pi(s_t)}$

GOAL

Find a **sequence of actions,** $\boldsymbol{\pi(s_t)}$**,** such that the resulting sequence of states *maximizes* the total discounted reward, ***U***

MARKOV'S DECISION PROCESS

$$U(s_0, \dots) = R(s_0) + \gamma^1 R(s_1) + \cdots + \gamma^N R(s_N) + \cdots$$

We move from s_t to s_{t+1} by **action** $\pi(s_t)$

GOAL

Find a **sequence of actions,** $\pi(s_t)$, such that the resulting sequence of states maximizes the total discounted reward, ***U***

Optimal Policy:

The policy π^* that *maximizes* the expected utility U of the sequence of states generated by π^* for all initial states s

MARKOV'S DECISION PROCESS

Question:

How do we compute the optimal policy π^*?

BELLMAN'S EQUATION

- If action $\boldsymbol{a}$ is taken in state $\boldsymbol{s}$,

$$\boldsymbol{U(s) = R(s,a) + \gamma \sum_{S'} T(s,a,s')\, U(s')}$$

$\boldsymbol{U(s)}$**:** Utility in state $\boldsymbol{s}$

$\boldsymbol{R(s,a)}$**:** Reward when taking action $\boldsymbol{a}$ from state $\boldsymbol{s}$

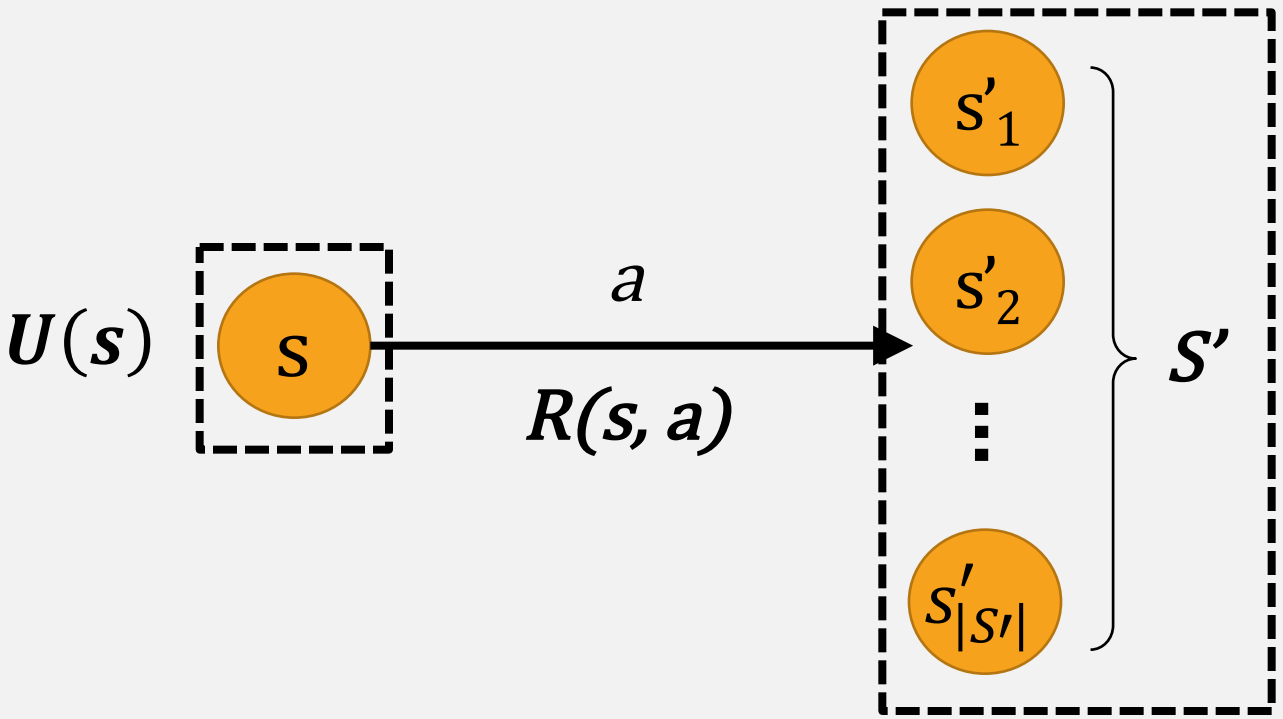
$\boldsymbol{\gamma}$**:** Discount factor

$\boldsymbol{T(s,a,s')}$: Transition probability from $\boldsymbol{s}$ to $\boldsymbol{s'}$ by taking $\boldsymbol{a}$

BELLMAN'S EQUATION

- If action $\boldsymbol{a}$ is taken in state $\boldsymbol{s}$,

$$\boldsymbol{U(s) = R(s,a) + \gamma \sum_{S'} T(s,a,s')\, U(s')}$$

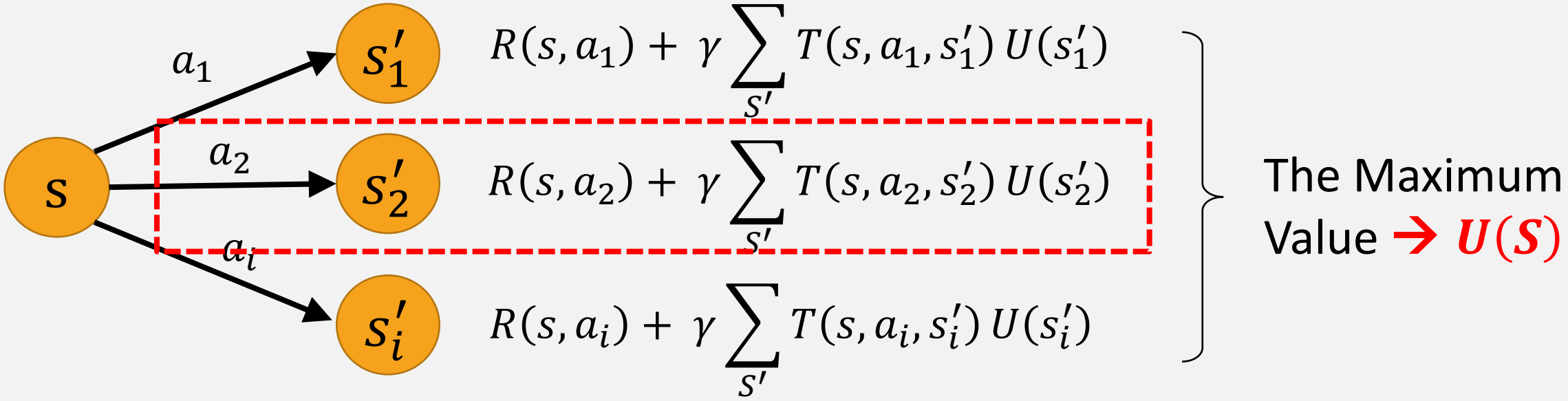


$$\boldsymbol{\gamma \sum_{S'} T(s,a,s')\, U(s')}$$

Expected value of future rewards from state s'

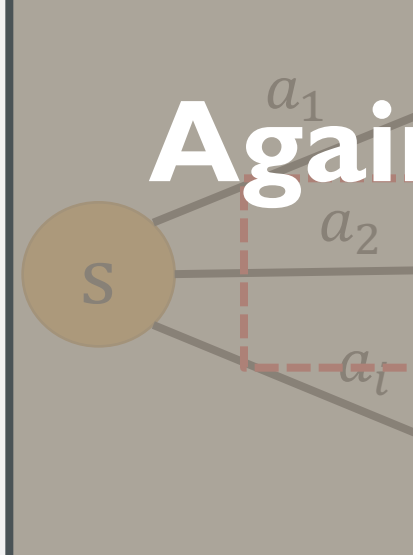
BELLMAN'S OPTIMALITY CRITERION

$$\boldsymbol{U(s) = \max_{a} \left(R(s,a) + \gamma \sum_{s'} T(s,a,s')\, U(s')\right)}$$



BELLMAN'S OPTIMALITY CRITERION

$$U(S) = \max_{a} \left(R(s,a) + \gamma \sum_{s'} T(s,a,s')\, U(s')\right)$$



$$R(s,a_1) + \gamma \sum_{s'} T(s,a_1,s'_1)\, U(s'_1)$$

$$R(s,a_2) + \gamma \sum_{s'} T(s,a_2,s'_2)\, U(s'_2)$$

$$R(s,a_i) + \gamma \sum_{s'} T(s,a_i,s'_i)\, U(s'_i)$$

The Maximum Value → $U(S)$

Again, how do we find the optimal policy?

POLICY ITERATION

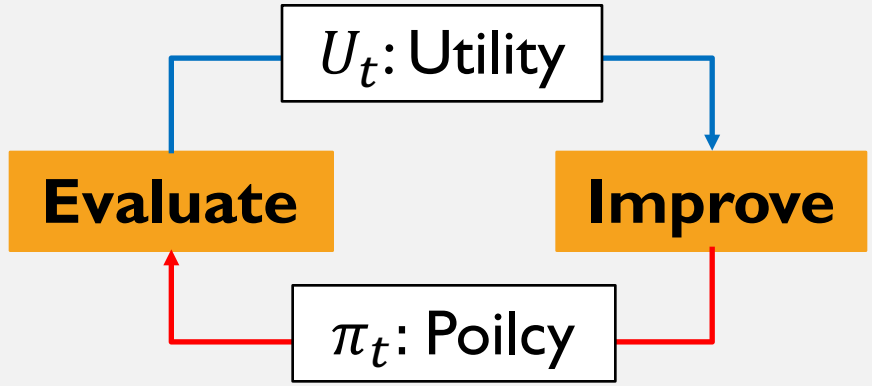
Initialize: $\boldsymbol{\pi_0} \leftarrow$ guess

Evaluate: given $\boldsymbol{\pi_t}$,

calculate $\boldsymbol{U_t(s)} = R(s) + \gamma \sum_{s'} T(s, \boldsymbol{\pi_t}(s), s')\, \boldsymbol{U_t(s')})$ (Bellman's eq)

Improve: $\boldsymbol{\pi_{t+1}} = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s, \boldsymbol{\pi_t}(s), s')\, \boldsymbol{U_t(s')}$)

POLICY ITERATION



Initialize: $\boldsymbol{\pi_0}$ ← guess

Evaluate: given $\boldsymbol{\pi_t}$,

calculate $\boldsymbol{U_t(s)} = R(s) + \gamma \sum_{s'} T(s, \boldsymbol{\pi_t}(s), s')\, \boldsymbol{U_t(s')})$ (Bellman's eq)

Improve: $\boldsymbol{\pi_{t+1}} = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s, \boldsymbol{\pi_t}(s), s')\, \boldsymbol{U_t(s')}$)

Repeat until the Utility converges → $\boldsymbol{U_t(s)} \approx \boldsymbol{U_{t+1}(s)}$

VALUE ITERATION

Initialize: Start with arbitrary utilities

Update utilities based on neighbors

$$\hat{\mathrm{U}}_{\boldsymbol{t+1}}(\boldsymbol{s}) = R(s) + \gamma \max_{a} \sum_{s'} T(s, \pi_{\boldsymbol{t}}(s), s')\, \hat{\mathrm{U}}_{\boldsymbol{t}}(\boldsymbol{s'})$$ (Bellman's eq)

VALUE ITERATION

Initialize: Start with arbitrary utilities

Update utilities based on neighbors

$$\hat{U}_{t+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, \pi_t(s), s') \hat{U}_t(s'))$$ (Bellman's eq)

→ Repeat until converges.

The quality of Utility gets better as more *truth* gets involved

CHECKPOINT

- **Markov Decision Process**

  Decision making process in a cognitive dynamic system

- **Bellman's Optimality Criterion**

  Defines the optimization problem the decision-maker must solve

- **Policy Iteration**

  Finds the optimal policy by iteratively estimating the policy

- **Value Iteration**

  Finds the optimal policy by iteratively estimating the utility

INTERMISSION

REINFORCEMENT LEARNING

as Dynamic Programming

REINFORCEMENT LEARNING RECAP

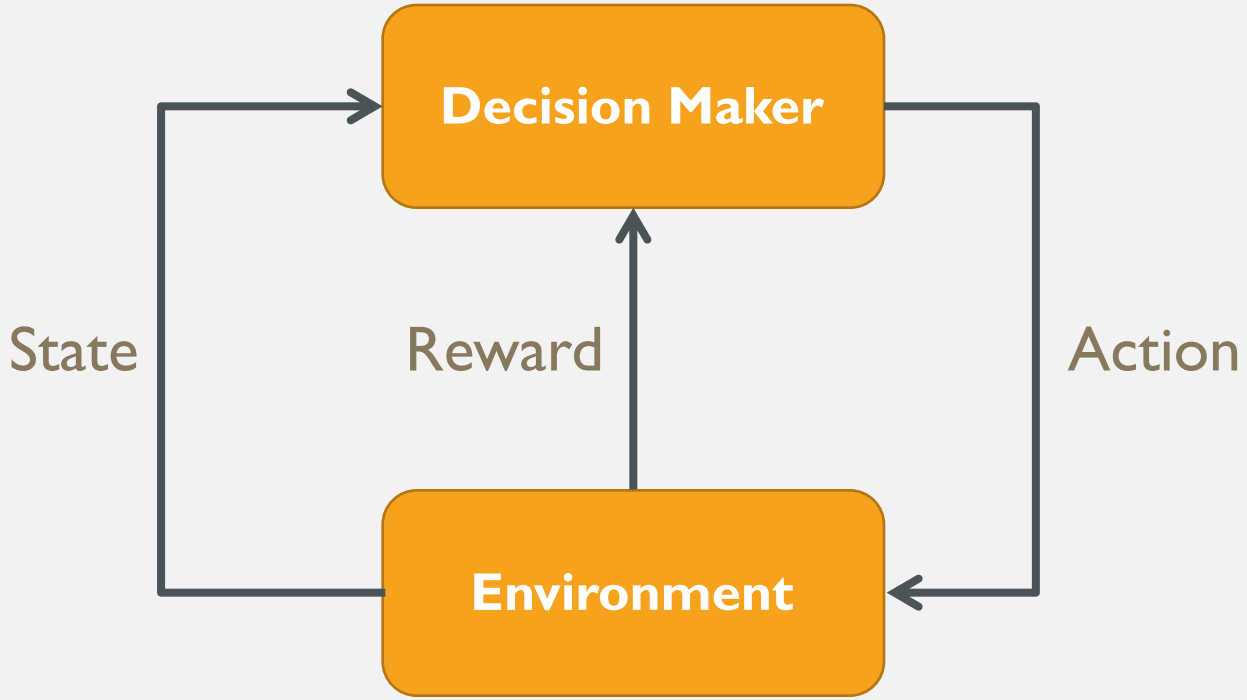
REINFORCEMENT LEARNING RECAP

Model-Based

Model-Free

REINFORCEMENT LEARNING RECAP

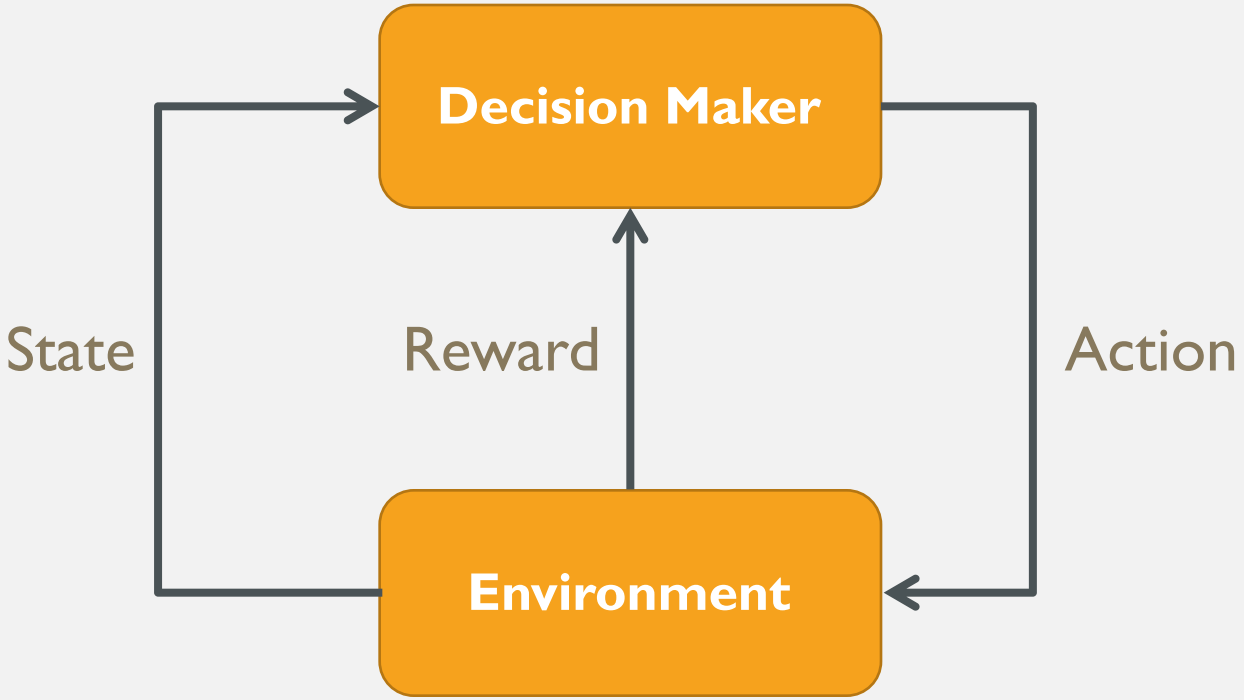
Model-Based (MDP)



- Set of States: $\boldsymbol{S} = \{\boldsymbol{s}\}$
- Set of Actions: $\boldsymbol{A} = \{\boldsymbol{a}\}, \boldsymbol{a}: \boldsymbol{S} \rightarrow \boldsymbol{S}$
- Reward Function: $\boldsymbol{R}(\boldsymbol{s})$
- Transition Model: $\boldsymbol{T}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')$ or $\boldsymbol{P}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$
- Policy: $\boldsymbol{\pi}(\boldsymbol{s}) \rightarrow \boldsymbol{a}$

REINFORCEMENT LEARNING RECAP

Model-Free



- Set of States: $\boldsymbol{S} = \{\boldsymbol{s}\}$
- Set of Actions: $\boldsymbol{A} = \{\boldsymbol{a}\}, \boldsymbol{a}: \boldsymbol{S} \rightarrow \boldsymbol{S}$

There is no explicit model for ***R*** and ***T***
We instead have explicit transition data:
$< \boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}', \boldsymbol{r} >$

- Policy: $\boldsymbol{\pi}(\boldsymbol{s}) \rightarrow \boldsymbol{a}$

MODEL-FREE REINFORCEMENT LEARNING

- Temporal Difference Learning
- Q-Learning

TEMPORAL DIFFERENCE (TD) LEARNING

$$\boldsymbol{U(s) = R(s,a) + \gamma \sum_{s'} T(s,a,s')\, U(s')}$$

Upon an action $a = \pi(s)$

For all s', successor of s, $U(s)$ must be "in between"

a) the new value considering only s': $R(s) + \gamma\, U(s')$

b) the old value $U(s)$

TEMPORAL DIFFERENCE (TD) LEARNING

$$\boldsymbol{U(s) = R(s,a) + \gamma \sum_{s'} T(s,a,s')\, U(s')}$$

Upon an action $a = \pi(s)$

For all s', successor of s, $U(s)$ must be "in between"

a) the new value considering only s': $R(s) + \gamma\, U(s')$

b) the old value $U(s)$

The Notion of Temporal Difference

TEMPORAL DIFFERENCE (TD) LEARNING

$$\boldsymbol{U(s) \coloneqq (1 - \alpha)\, U(s) \; + \; \alpha\, (R(s) \; + \; \gamma\, U(s'))}$$

The new approximation of $\boldsymbol{U(s)}$ using $\boldsymbol{\alpha}$
when moving from a state $\boldsymbol{s}$ to another state $\boldsymbol{s'}$

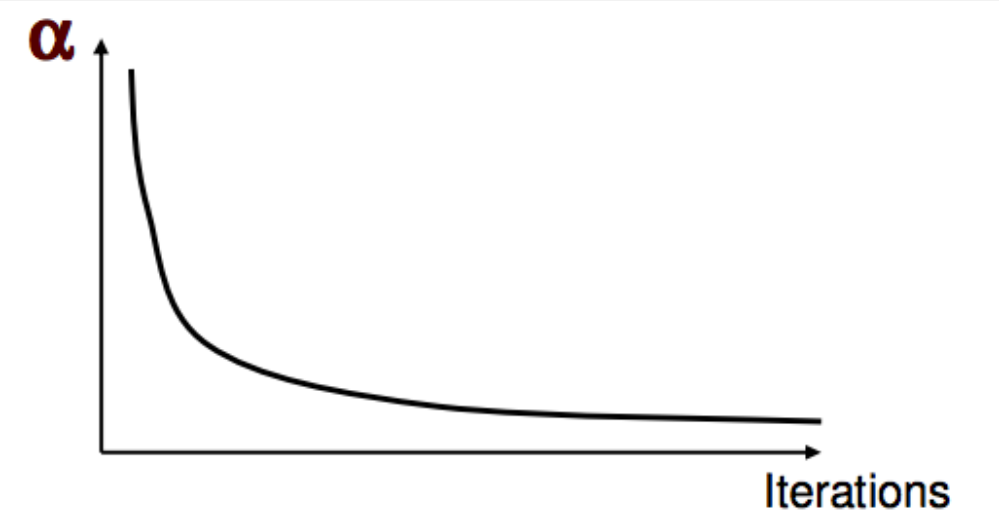
Rearrange the above equation to update $\boldsymbol{U(s)}$**:**

$$\boldsymbol{U(s) \coloneqq U(s) \; + \; \alpha\, (R(s) \; + \; \gamma\, U(s') - \; U(s))}$$

TEMPORAL DIFFERENCE (TD) LEARNING

$$U(s) := U(s) + \alpha(R(s) + \gamma U(s') - U(s))$$

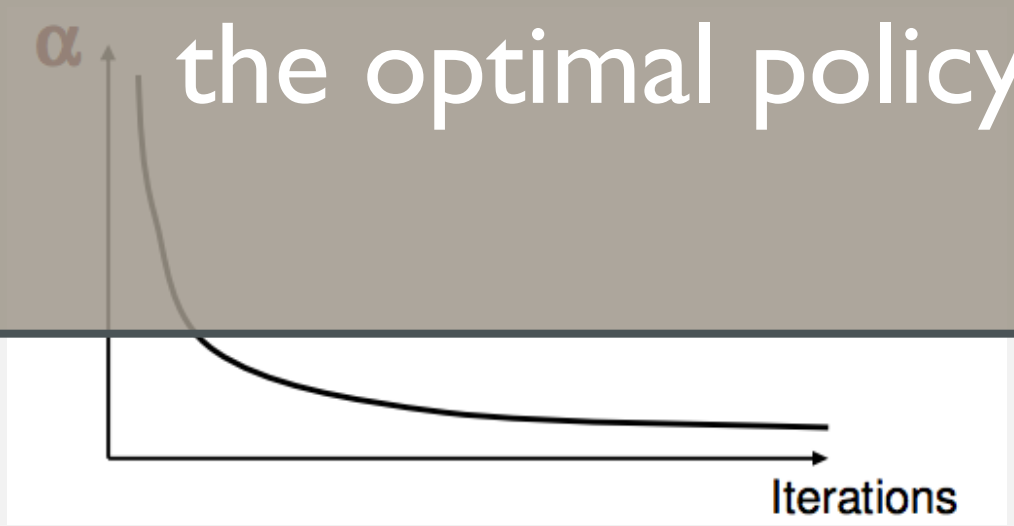
Learning Rate: $0 \leq \alpha < 1$



TEMPORAL DIFFERENCE (TD) LEARNING

$$U(s) := U(s) + \alpha(R(s) + \gamma U(s') - U(s))$$

But this says nothing about the optimal policy, π^*



Q-LEARNING

$$\boldsymbol{U(s) \coloneqq U(s) \; + \; \alpha \, (R(s) \; + \; \gamma \, U(s') - \; U(s))}$$

Policy is about taking an action $\boldsymbol{a}$ from a state $\boldsymbol{s}$
--> Let's introduce action to $\boldsymbol{U(s)}$!

$\boldsymbol{Q(s, a)}$**:** Expected sum of future discounted rewards
after taking action $\boldsymbol{a}$ at state **s**

$$\boldsymbol{Q(s, a) \coloneqq Q(s, a) + \alpha(R(s) + \gamma \; \max_{a'} Q(s', a') - Q(s, a))}$$

Q-LEARNING

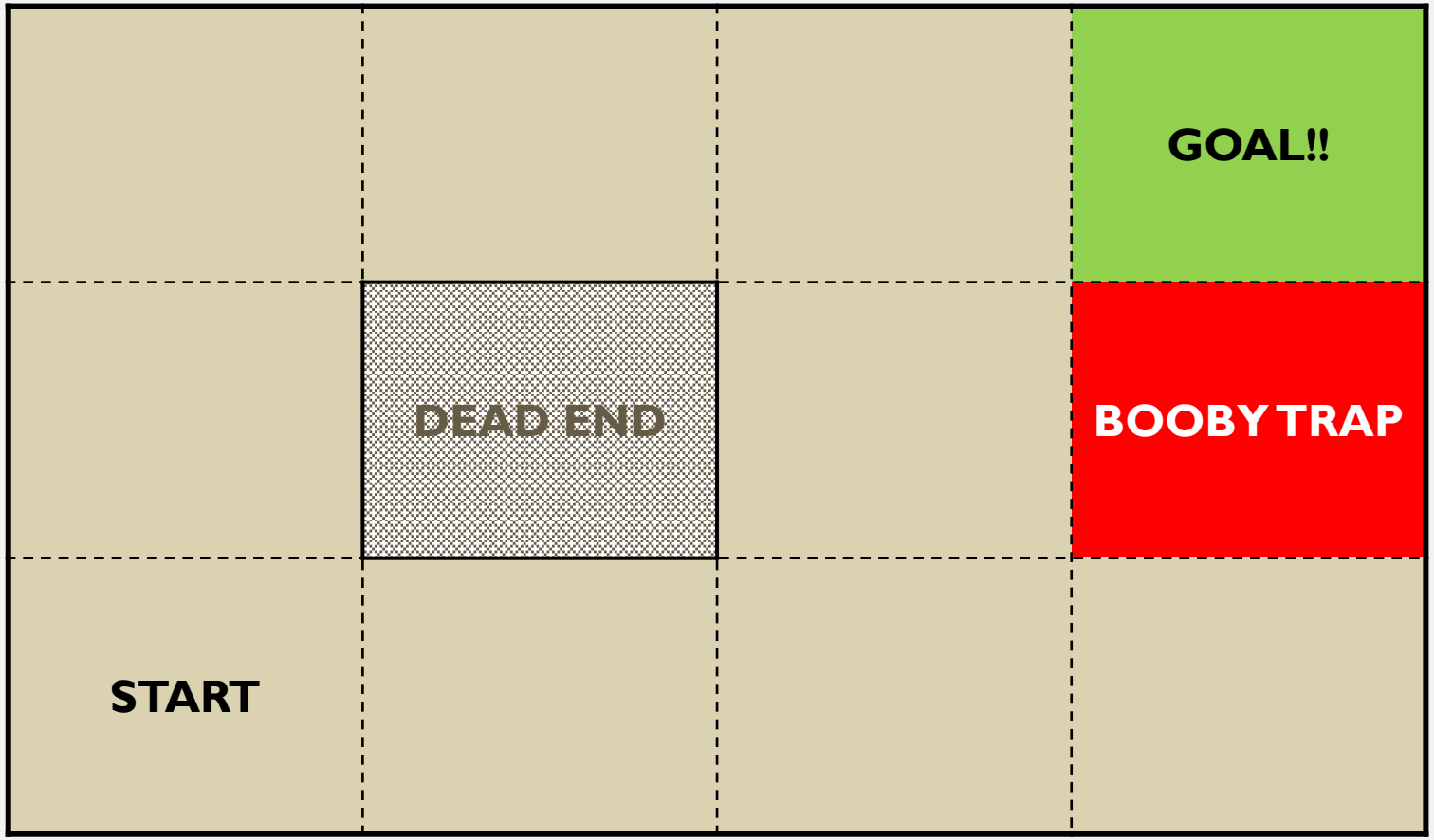
$$Q(s,a) \coloneqq Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Finding the optimal policy is now a matter of finding the actions for the states in maximizing the Q-value:

$$\pi(s) = \operatorname*{argmax}_{a} Q(s,a)$$

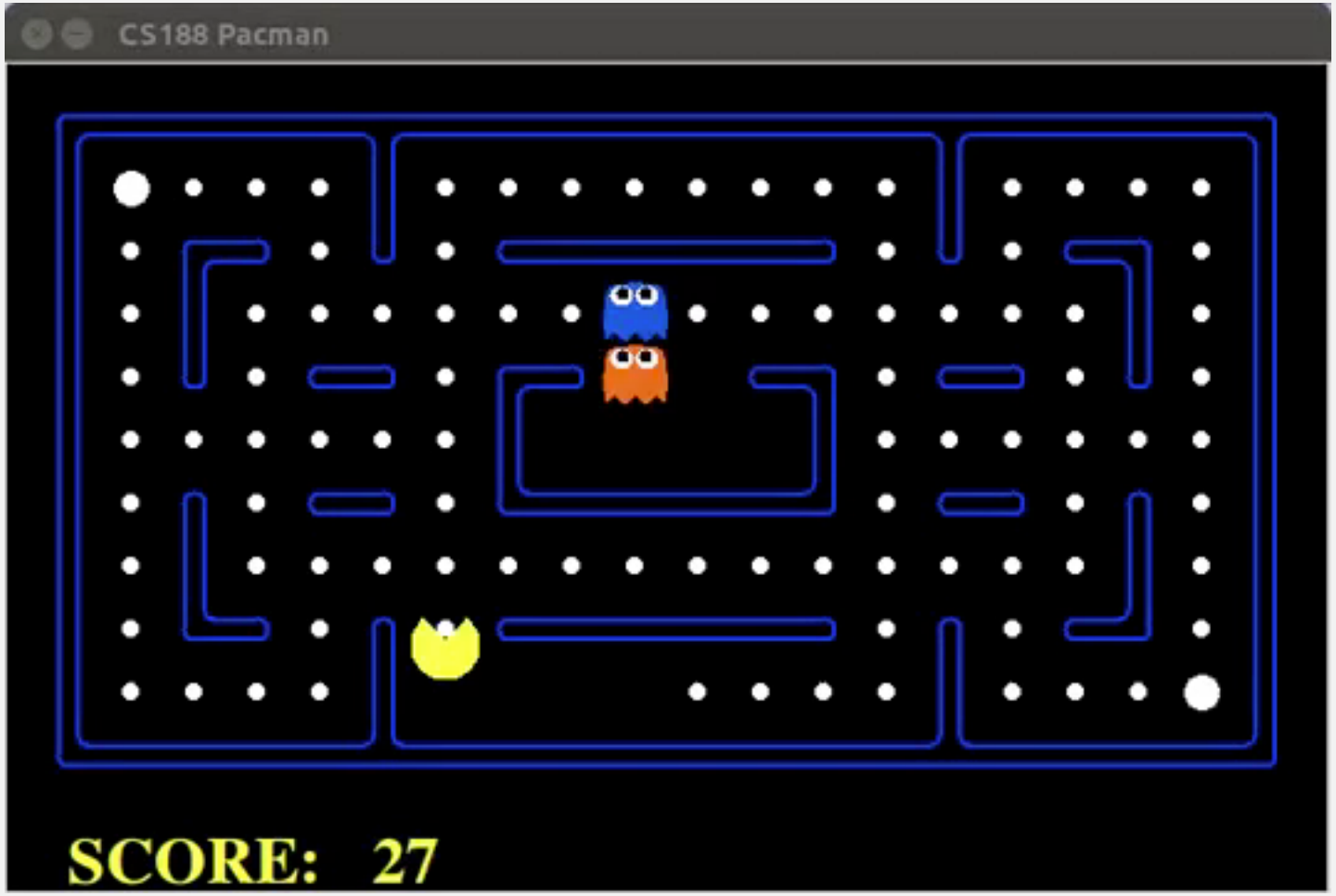
EXAMPLE

SCENARIO



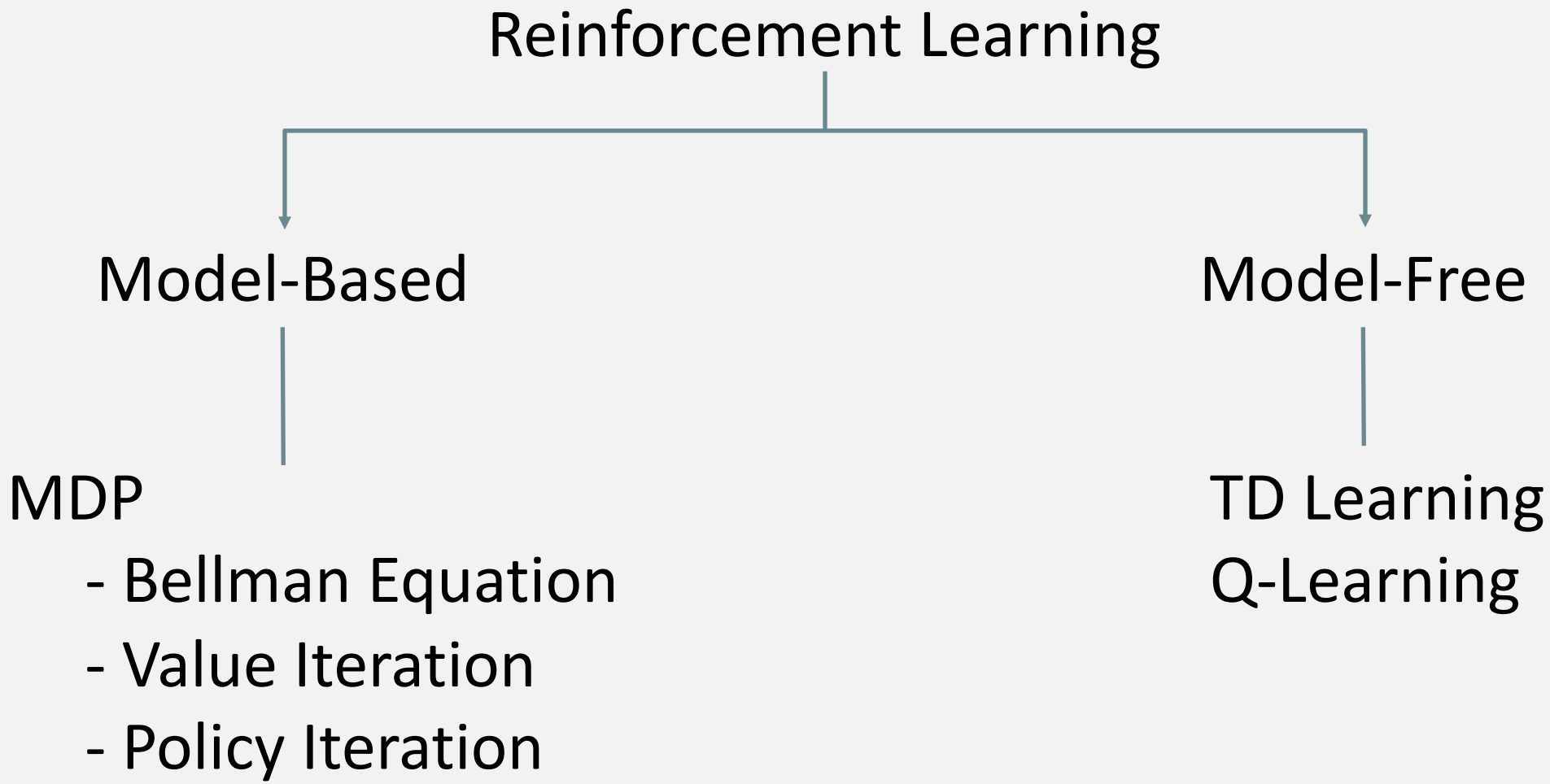
You can move UP, DOWN, LEFT, RIGHT

PACMAN



SUMMARY

SUMMARY


Reinforcement Learning
Model-Based
Model-Free
MDP
- Bellman Equation
- Value Iteration
- Policy Iteration
TD Learning
Q-Learning

DISCUSSION

QUESTIONS FOR THOUGHT

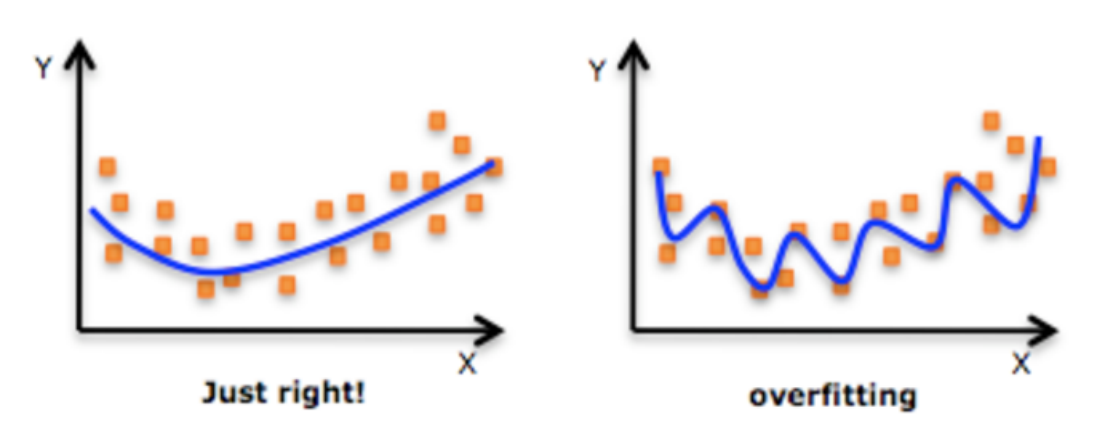
Q1. What if the problem we are trying to solve is large scale?

$\Rightarrow$ State-space is large!

$\Rightarrow$ ex. value iteration may take forever!

$\Rightarrow$ examining the entire training dataset is endless!

Q2. What if our solution "overfits" data given by the environment?



APPROXIMATE DYNAMIC PROGRAMMING

LINEAR APPROXIMATION APPROACH

The *curse-of-dimensionality problem*: the exponential growth of computational complexity with increasing dimensionality of the state space.

a) Abandon the idealized notion of optimality
and be content with a suboptimal solution ☺

b) Approximate functions for data: $< s, a, s', r >$
- can prevent overfitting

Q & A

APPENDIX

NOTATION & TERMINOLOGY

- We found that the notations used in the book are tricky to understand, and concluded to use the equivalent alternatives that were easier.

<TEXTBOOK>

- $i \sim X_n$: State
- $\mu(i)$: Policy
- $J^{\pi}\ (i)$: Cost-to-go function
- $g\left(X_n, \mu_{n,}\, X_{n+1}\right)$: Transition Cost
- $p_{ij}(\mu(i))$: Transition Probability

<THE SLIDES>

- $s \in S$: State
- $\pi(s)$: Policy
- $U(s)$: Utility
- $R(s, a)$: Reward (or R(s))
- $T(s, a, s')$: Transition Probability